METHOD AND APPARATUS FOR UPDATING AND INVALIDATING STORE DATA

Cross Reference To Related Application(s)

This is a divisional of a copending application serial number 10/230,188, entitled "METHOD AND APPARATUS FOR UPDATING AND INVALIDATING STORE DATA," filed on August 29, 2002, now U.S. Patent No. 6,772,316, which in turn is a divisional of application serial number 09/466,306 entitled "UPDATING AND INVALIDATING STORE DATA AND REMOVING STALE CACHE LINES IN A PREVALIDATED TAG CACHE DESIGN," now U.S. Patent 6,470,437. These applications and the patent are hereby incorporated by reference.

DT 2/4/05

Technical Field

The technical field encompasses computer systems employing prevalidated cache tag designs. In particular, the technical field encompasses designs to support store updates and invalidates and removal of stale cache lines out of a cache.

Background

Computer systems may employ a multi-level hierarchy of memory, with relatively fast, expensive but limited-capacity memory at the highest level of the hierarchy and proceeding to relatively slower, lower cost but higher-capacity memory at the lowest level of the hierarchy. The hierarchy may include a small fast memory called a cache, either physically integrated within a processor or mounted physically close to the processor for speed. The computer system may employ separate instruction caches and data caches. In addition, the computer system may use multiple levels of caches. The use of a cache is transparent to a computer program at the instruction level and can thus be added to a computer architecture without changing the instruction set or requiring modification to existing programs.

A cache hit occurs when a processor requests an item from a cache and the item is present in the cache. A cache miss occurs when a processor requests an item from a cache and the item is not present in the cache. In the event of a cache miss, the processor retrieves the requested item from a lower level of the memory hierarchy. In many processor designs, the time required to access an item for a cache hit is one of the primary limiters for the clock rate of the processor if the designer is seeking a single cycle cache access time. In other designs, the cache access time may be multiple cycles, but the performance of a processor can be improved in most cases when the cache access time in

DT 2/4/05

now, U.S. Patent No. 6,014,732)

entitled CACHE MEMORY WITH REDUCED ACCESS TIME, the disclosure of which is hereby incorporated by reference.

The micro-architecture illustrated in Figure 3 includes a prevalidated tag cache. The prevalidation imposes restrictions on how the TLBs in the micro-architecture work if the computer micro-architecture is designed to maximize overall bandwidth while minimizing cache load latency. The prevalidated tag cache, for example, provides very fast access time for certain loads but the micro-architecture may be designed to restrict the translations between virtual and physical address and restrict the distribution of processing among the different cache levels. The micro-architecture may provide for fast integer loads and a high bandwidth for floating point loads, for example. That is, integer load data needs to have fast access timing but its working set size is generally small. To optimize integer load latency, some processors provide a small but fast first level cache. To provide virtual address translation and avoid address aliasing problems, some processors must access the TLB to provide a physical address for checking with the cache tags to determine if the data is present in the cache or not. To decrease the memory latency for fast integer data access, TLB size may be limited to a small number of entries (such as 16 to 32). This conflicts with the large number of entries required on processors with large cache structures that could require 256 or more TLB entries.

In a prevalidated cache tag system, such as that shown in Figure 3, the TLB entries are logically used in the cache tag to identify the cache lines. When a TLB entry is removed, control is normally used to invalidate all the data in the prevalidated cache tag that is associated with the removed TLB entry. However, this action may slow processing since one TLB entry may map to much or all of the data cache. The TLB may then be continually swapping pages in and out of memory (i.e., thrashing) instead of supporting program execution.

Floating point data processing performance is usually limited by the memory bandwidth in and out of the floating point execution units. As opposed to integer load data accesses, which need a low latency, floating point accesses can usually be scheduled and can therefore endure a longer latency period. Likewise, while the integer data size is usually small, floating point data sets are usually very large. Ideally, TLB operations for floating point load/store operations will provide both high bandwidth and large data space translations (large number of TLB entries accessed). One design provides full bandwidth for all memory ports and a large but slower TLB for translation of floating point requests.